Ramdeobaba University, Nagpur

Department of Computer Science and Engineering

Session: 2025-26

DAA LAB **III Semester**

Name: Vardhan Ingole
Section: A4
Batch: B3
Roll No: 43

Practical -5 (Part 1)

Aim: Implement a dynamic algorithm for Longest Common Subsequence (LCS) to find the length and LCS for DNA sequences.

Problem Statement:

(i) DNA sequences can be viewed as strings of A, C, G, and T characters, which
represent nucleotides. Finding the similarities between two DNA sequences are an
important computation performed in bioinformatics.

[Note that a subsequence might not include consecutive elements of the original sequence.]

TASK 1: Find the similarity between the given X and Y sequence.
X=AGCCCTAAGGGCTACCTAGCTT
Y= GACAGCCTACAAGCGTTAGCTTG

Code:

```c
#include <stdio.h>
#include <string.h>

#define MAX 100

void printMatrix(int dp[MAX][MAX], char dir[MAX][MAX], int m, int n, char
X[], char Y[]) {
    printf("\n Cost Matrix with Directions:\n\n   ");
    for (int j = 0; j <= n; j++) {
        if (j == 0) printf("    ");
        else printf("  %c ", Y[j-1]);
    }
    printf("\n");

    for (int i = 0; i <= m; i++) {
        if (i == 0) printf("   ");
        else printf(" %c ", X[i-1]);

        for (int j = 0; j <= n; j++) {
            printf("%2d%c ", dp[i][j], dir[i][j]);
        }
        printf("\n");
    }
}

void printLCS(char X[], char Y[], int m, int n, int dp[MAX][MAX]) {
    int i = m, j = n;
    char lcs[MAX];
    int index = dp[m][n];
    lcs[index] = '\0';

    while (i > 0 && j > 0) {
        if (X[i - 1] == Y[j - 1]) {
            lcs[index - 1] = X[i - 1];
            i--;
            j--;
            index--;
        }
        else if (dp[i - 1][j] > dp[i][j - 1]) {
            i--;
```

```
        }
        else {
            j--;
        }
    }
    printf("\nLongest Common Subsequence: %s\n", lcs);
}

int LCS_with_path(char X[], char Y[], int m, int n) {
    int dp[MAX][MAX];
    char dir[MAX][MAX];
    int i, j;

    for (i = 0; i <= m; i++) {
        for (j = 0; j <= n; j++) {
            dp[i][j] = 0;
            dir[i][j] = ' ';
        }
    }

    for (i = 1; i <= m; i++) {
        for (j = 1; j <= n; j++) {
            if (X[i - 1] == Y[j - 1]) {
                dp[i][j] = dp[i - 1][j - 1] + 1;
                dir[i][j] = '\\';
            } else {
                if (dp[i - 1][j] >= dp[i][j - 1]) {
                    dp[i][j] = dp[i - 1][j];
                    dir[i][j] = '|';
                } else {
                    dp[i][j] = dp[i][j - 1];
                    dir[i][j] = '-';
                }
            }
        }
    }

    printf("\nLength of LCS: %d\n", dp[m][n]);
    printMatrix(dp, dir, m, n, X, Y);
    printLCS(X, Y, m, n, dp);
```

```
    return dp[m][n];
}

int main() {
    char X[] = "AGCCCTAAGGGCTACCTAGCTT";
    char Y[] = "GACAGCCTACAAGCGTTAGCTTG";
    int m = strlen(X);
    int n = strlen(Y);

    LCS_with_path(X, Y, m, n);
    return 0;
}
```

Output:

```
Length of LCS: 16

Cost Matrix with Directions:

   |  |G  A  C  A  G  C  C  T  A  C  A  A  G  C  G  T  T  A  G  C  T  T  G
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
A  0  0| 1\ 1- 1\ 1- 1- 1- 1- 1\ 1- 1\ 1\ 1- 1- 1- 1- 1- 1\ 1- 1- 1- 1- 1-
G  0  1\ 1| 1| 1| 2\ 2- 2- 2- 2- 2- 2- 2- 2\ 2- 2\ 2- 2- 2- 2\ 2- 2- 2- 2\
C  0  1| 1| 2\ 2- 2| 3\ 3\ 3- 3- 3\ 3- 3- 3- 3\ 3- 3- 3- 3- 3- 3\ 3- 3- 3-
C  0  1| 1| 2\ 2| 2| 3\ 4\ 4- 4- 4\ 4- 4- 4- 4\ 4- 4- 4- 4- 4- 4\ 4- 4- 4-
C  0  1| 1| 2\ 2| 2| 3\ 4\ 4| 4| 5\ 5- 5- 5- 5\ 5- 5- 5- 5- 5- 5\ 5- 5- 5-
T  0  1| 1| 2| 2| 2| 3| 4| 5\ 5- 5| 5| 5| 5| 5| 5| 6\ 6\ 6- 6- 6- 6\ 6\ 6-
A  0  1| 2\ 2| 3\ 3- 3| 4| 5| 6\ 6- 6\ 6\ 6- 6- 6- 6| 6| 7\ 7- 7- 7- 7- 7-
A  0  1| 2\ 2| 3\ 3| 3| 4| 5| 6\ 6| 7\ 7\ 7- 7- 7- 7- 7- 7\ 7| 7| 7| 7| 7|
G  0  1\ 2| 2| 3| 4\ 4- 4| 5| 6| 6| 7| 7| 8\ 8- 8\ 8- 8- 8- 8\ 8- 8- 8- 8\
G  0  1\ 2| 2| 3| 4\ 4| 4| 5| 6| 6| 7| 7| 8\ 8| 9\ 9- 9- 9- 9\ 9- 9- 9- 9\
G  0  1\ 2| 2| 3| 4\ 4| 4| 5| 6| 6| 7| 7| 8\ 8| 9\ 9| 9| 9| 10\ 10- 10- 10- 10\
C  0  1| 2| 3\ 3| 4| 5\ 5\ 5| 6| 7\ 7| 7| 8| 9\ 9| 9| 9| 9| 10| 11\ 11- 11- 11-
T  0  1| 2| 3| 3| 4| 5| 5| 6\ 6| 7| 7| 7| 8| 9| 9| 10\ 10\ 10- 10| 11| 12\ 12\ 12-
A  0  1| 2\ 3| 4\ 4| 5| 5| 6| 7\ 7| 8\ 8\ 8| 9| 9| 10| 10| 11\ 11- 11| 12| 12| 12|
C  0  1| 2| 3\ 4| 4| 5\ 6\ 6| 7| 8\ 8| 8| 8| 9\ 9| 10| 10| 11| 11| 12\ 12| 12| 12|
C  0  1| 2| 3\ 4| 4| 5\ 6\ 6| 7| 8\ 8| 8| 8| 9\ 9| 10| 10| 11| 11| 12\ 12| 12| 12|
T  0  1| 2| 3| 4| 4| 5| 6| 7\ 7| 8| 8| 8| 8| 9| 9| 10\ 11\ 11| 11| 12| 13\ 13\ 13-
A  0  1| 2\ 3| 4\ 4| 5| 6| 7| 8\ 8| 9\ 9\ 9- 9| 9| 10| 11| 12\ 12- 12| 13| 13| 13|
G  0  1\ 2| 3| 4| 5\ 5| 6| 7| 8| 8| 9| 9| 10\ 10- 10\ 10| 11| 12| 13\ 13- 13| 13| 14\
C  0  1| 2| 3\ 4| 5| 6\ 6\ 7| 8| 9\ 9| 9| 10| 11\ 11- 11- 11| 12| 13| 14\ 14- 14- 14|
T  0  1| 2| 3| 4| 5| 6| 6| 7\ 8| 9| 9| 9| 10| 11| 11| 12\ 12\ 12| 13| 14| 15\ 15\ 15-
T  0  1| 2| 3| 4| 5| 6| 6| 7\ 8| 9| 9| 9| 10| 11| 11| 12\ 13\ 13- 13| 14| 15\ 16\ 16-

Longest Common Subsequence: GCCCTAAGCTTAGCTT

[Done] exited with code=0 in 0.478 seconds
```

Practical -5 (Part 2)

TASK-2:

Find the longest repeating subsequence (LRS). Consider it as a variation of the longest common subsequence (LCS) problem.

Let the given string be S. You need to find the LRS within S. To use the LCS framework, you effectively compare S with itself. So, consider string1 = S and string2 = S.

Example:

AABCBDC

LRS= ABC or ABD

Code:

```
#include <stdio.h>
#include <string.h>

char lrs_results[100][50];
int lrs_count = 0;

void reverse_string(char *str) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char temp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = temp;
    }
}

void find_all_lrs(char* str, int i, int j, char* current_lrs, int
current_len, int dp[][100]) {
    if (i == 0 || j == 0) {
        if (current_len > 0) {
            current_lrs[current_len] = '\0';
            reverse_string(current_lrs);

            int is_duplicate = 0;
            for (int k = 0; k < lrs_count; k++) {
                if (strcmp(lrs_results[k], current_lrs) == 0) {
                    is_duplicate = 1;
```

```
                    break;
                }
            }

            if (!is_duplicate) {
                strcpy(lrs_results[lrs_count], current_lrs);
                lrs_count++;
            }

            reverse_string(current_lrs);
        }
        return;
    }

    if (str[i - 1] == str[j - 1] && i != j) {
        current_lrs[current_len] = str[i - 1];
        find_all_lrs(str, i - 1, j - 1, current_lrs, current_len + 1, dp);
    } else {
        if (dp[i - 1][j] > dp[i][j - 1]) {
            find_all_lrs(str, i - 1, j, current_lrs, current_len, dp);
        } else if (dp[i][j - 1] > dp[i - 1][j]) {
            find_all_lrs(str, i, j - 1, current_lrs, current_len, dp);
        } else {
            find_all_lrs(str, i - 1, j, current_lrs, current_len, dp);
            find_all_lrs(str, i, j - 1, current_lrs, current_len, dp);
        }
    }
}

int main() {
    char str[] = "AABCBDC";
    int n = strlen(str);
    int dp[100][100];

    // Initialize DP table
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= n; j++) {
            dp[i][j] = 0;
        }
    }
```

```
    // Fill DP table for LRS
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (str[i - 1] == str[j - 1] && i != j) {
                dp[i][j] = 1 + dp[i - 1][j - 1];
            } else {
                dp[i][j] = (dp[i - 1][j] > dp[i][j - 1]) ? dp[i - 1][j] :
dp[i][j - 1];
            }
        }
    }

    printf("String: %s\n", str);
    printf("Length of LRS: %d\n", dp[n][n]);

    char lrs_str[50];
    find_all_lrs(str, n, n, lrs_str, 0, dp);

    printf("All possible LRSs:\n");
    for (int i = 0; i < lrs_count; i++) {
        printf("%s\n", lrs_results[i]);
    }

    return 0;
}
```

Output:

```
String: AABCBDC
Length of LRS: 3
All possible LRSs:
ABC
ABA
PS C:\Users\DT USER\Desktop\output>
```